LA-UR-21-22897
Approved for public release; distribution is unlimited.

Title: MATAR: Data-Oriented Sparse Data Representation

Author(s): Tafolla, Tanya
Nelluvelil, Eappen Sebastian
Moore, Jacob Linley
Dunning, Daniel Jeffrey
Morgan, Nathaniel Ray
Robey, Robert W.

Intended for: SIAM Conference on Computational Science and Engineering (Virtual)

Issued: 2021-03-24

MATAR: Data-Oriented Sparse Data Representation

Tanya V. Tafolla[1] , Eappen Nelluvelil[4], Jacob Moore[2], Daniel Dunning[5], Nathaniel Morgan[3], Robert Robey[3]

[1]University of California, Merced
[2]Mississippi State University
[3]Los Alamos National Laboratory
[4]Rice University
[5]Texas Tech University

MS 30: Software Productivity and Sustainability in CSE
SIAM Conference on Computational Science and Engineering
March 1, 2021

Change in Performance

There are two fundamental aspects of a simulation: the movement of data from memory, and the computation.
Both or one of these aspects will be a limiting factor for the simulation.

Change in Performance

There are two fundamental aspects of a simulation: the movement of data from memory, and the computation.
Both or one of these aspects will be a limiting factor for the simulation.
FLOPs or memory bound.

- FLOPs limitations can be addressed with more powerful architecture - hardware issue.
- Memory bound limitations are harder to address because one needs to be more data conscious.

Change in Performance

There are two fundamental aspects of a simulation: the movement of data from memory, and the computation.
Both or one of these aspects will be a limiting factor for the simulation.
FLOPs or memory bound.

- FLOPs limitations can be addressed with more powerful architecture - hardware issue.
- Memory bound limitations are harder to address because one needs to be more data conscious.

Performance Focus

Focus on *feeds* instead of *speeds*.

Change in Performance

There are two fundamental aspects of a simulation: the movement of data from memory, and the computation.
Both or one of these aspects will be a limiting factor for the simulation.
FLOPs or memory bound.

- FLOPs limitations can be addressed with more powerful architecture - hardware issue.
- Memory bound limitations are harder to address because one needs to be more data conscious.

Performance Focus

Focus on *feeds* instead of *speeds*.

- Feeds: ability to get the data to the processor.
- Speeds: time it takes to execute a FLOP or other basic computer instruction.

Goal

Develop a C++ library of data-oriented sparse data representations that is performant and portable across different architectures.

Data-Oriented Design

Data-Oriented Design:

- Shift the focus from *objects* to *data*.

Data-Oriented Design

Data-Oriented Design:

- Shift the focus from *objects* to *data*.
- What that means for us: memory layout pattern matches the access pattern.

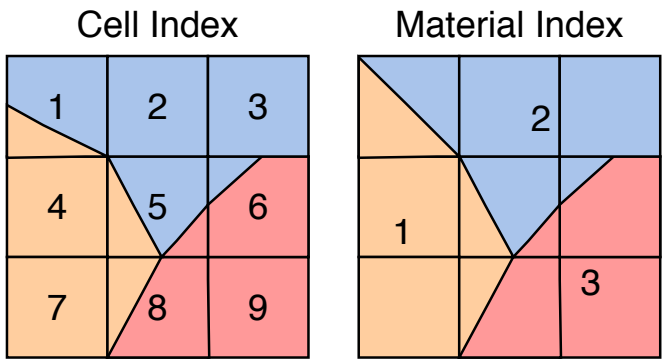
Data-Oriented Design

Data-Oriented Design:

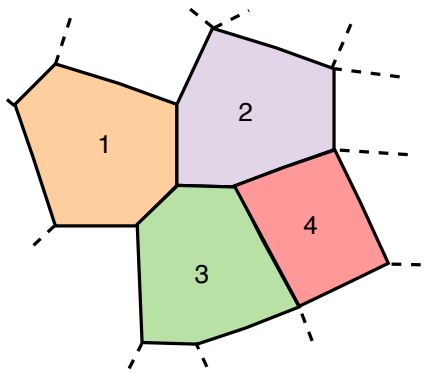
- Shift the focus from *objects* to *data*.
- What that means for us: memory layout pattern matches the access pattern.
- Layout = access, then less time is spend loading the data from main memory; data that is needed for *next* computation is already in cache.

The Need for Sparse Representation in Scientific Computing

Where do we need sparse data storage?



In some applications there can be multiple materials per cell and for Eulerian methods, the material moves through the cell. This results in a ragged memory layout either cell centric of material centric. Details in [6].



Polytopal meshes where the each cell has varying connectivity points to represent each cell.

The Need for Sparse Representation in Scientific Computing

Why address sparse data and **How to**:

- Providing data structures that map problem data efficiently in memory will yield better performance. $\rightarrow$ Forcing contiguous memory layout and access patterns.

The Need for Sparse Representation in Scientific Computing

Why address sparse data and **How to**:

- Providing data structures that map problem data efficiently in memory will yield better performance. $\rightarrow$ Forcing contiguous memory layout and access patterns.
- Commonly used sparse data representations, such as linked list, are slow and not easily parallelized. $\rightarrow$ Developed dynamic and static ragged arrays.

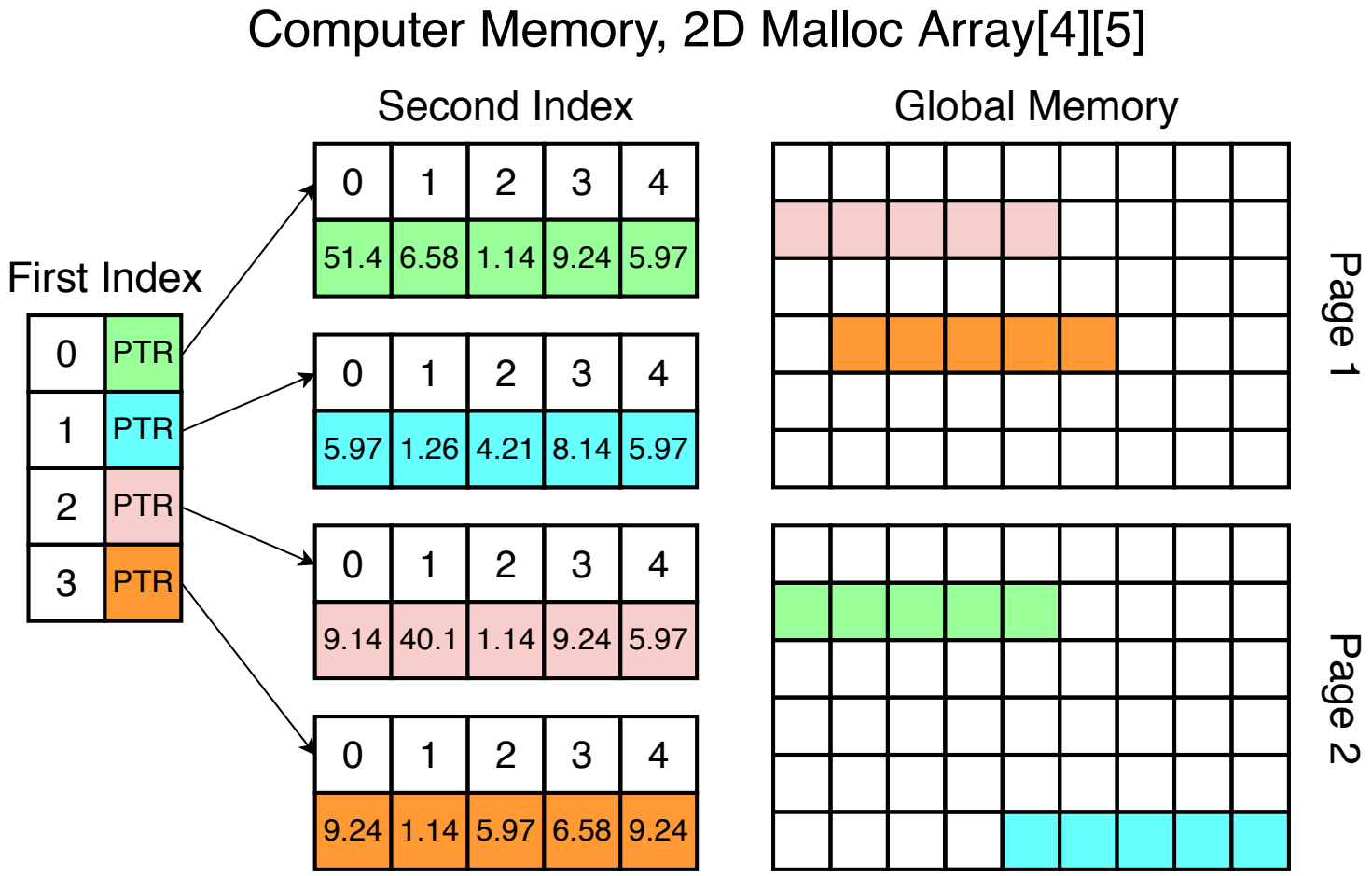
The Need for Sparse Representation in Scientific Computing

Why address sparse data and **How to**:

- Providing data structures that map problem data efficiently in memory will yield better performance. $\rightarrow$ Forcing contiguous memory layout and access patterns.
- Commonly used sparse data representations, such as linked list, are slow and not easily parallelized. $\rightarrow$ Developed dynamic and static ragged arrays.
- Want to run simulations on multiple CPU and GPU architectures, therefore the data needs to be portable without sacrificing performance. $\rightarrow$ GPU version of sparse data structures using `Kokkos`.

Current Limitations of Existing Frameworks

Limitations of Current Data Structures: Memory Layout



Using `malloc` and `new` in `C` and `C++`, respectively, does not necessarily result in contiguous allocated memory. Having the data scattered through global memory will lead to performance issues as more calls to main memory need to be made in order to obtain necessary data.

Other Performance Libraries:

1. Boost:
   - Provides multi-dimensional array class templates for C++ (*n*-dimensional contiguous arrays).
   - Other great features: sub-views, memory layout style, how to index arrays, reshape and resize.

Other Performance Libraries:

1. Boost:
   - Provides multi-dimensional array class templates for C++ (*n*-dimensional contiguous arrays).
   - Other great features: sub-views, memory layout style, how to index arrays, reshape and resize.
2. Matrix Template Library: Provides flexible multi-dimensional data structures for numerical solvers.

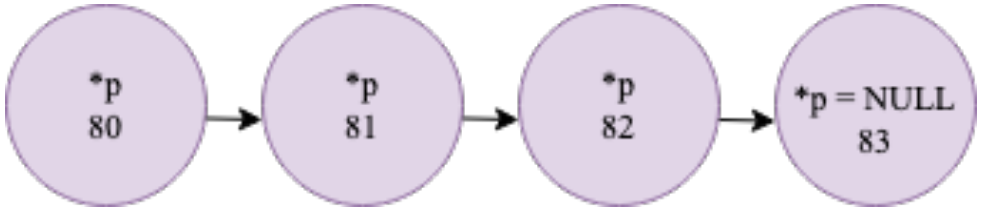
However, each of these tools have a targeted purposes. They focus on CPU application space with regular multi-dimensional arrays.

Other Performance Libraries:

1. Boost:
   - Provides multi-dimensional array class templates for C++ (*n*-dimensional contiguous arrays).
   - Other great features: sub-views, memory layout style, how to index arrays, reshape and resize.
2. Matrix Template Library: Provides flexible multi-dimensional data structures for numerical solvers.
3. Warwick Data Store:
   - Provides multi-dimensional array capabilities with a focus of letting programmers write data structures.
   - Collection of 3 classes: controller (high-level functionality to programmers), variable (allocate and manage memory), view (quick and easy access to underlying memory for data structure).

However, each of these tools have a targeted purposes. They focus on CPU application space with regular multi-dimensional arrays.

Sparse Data Addressed: Linked List



1. Pros:
   - add/remove elements anywhere throughout the list.
2. Cons:
   - Have to traverse the entire list to access an element.
   - Repeatedly adding/removing elements can cause re-allocations of heap memory.
   - Not guaranteed to be contiguous in memory.
   - Can not be easily ported to GPU; how do we keep track of all the pointers?

Introduction to MATAR

MATAR

To tackle the sparse representation, we develop MATAR.
A C++ library of contiguous, data-oriented **mat**rices and **ar**rays.

MATAR

To tackle the sparse representation, we develop MATAR.
A C++ library of contiguous, data-oriented **mat**rices and **ar**rays.
MATAR addresses:

- Performant: contiguous dense and sparse data representation.
- Portable: across CPU and GPUs (use Kokkos for GPU).
- Productivity: easy to create,use and integrate. User does not worry about memory management.

MATAR Data Structures Overview

Dense Data Representation:

		Indexing pattern	
		0-indexed	1-indexed
Access pattern	Column major	FArray ViewFArray	FMatrix ViewFMatrix
	Row major	CArray ViewCArray	CMatrix ViewCMatrix

Sparse Data Representation:

		Indexing pattern
		0-indexed
Access pattern	Column major	RaggedDownArray DynamicRaggedDownArray SparseColArray
	Row major	RaggedRightArray DynamicRaggedRightArray SparseRowArray

MATAR Sparse Data Structure: Ragged-Right

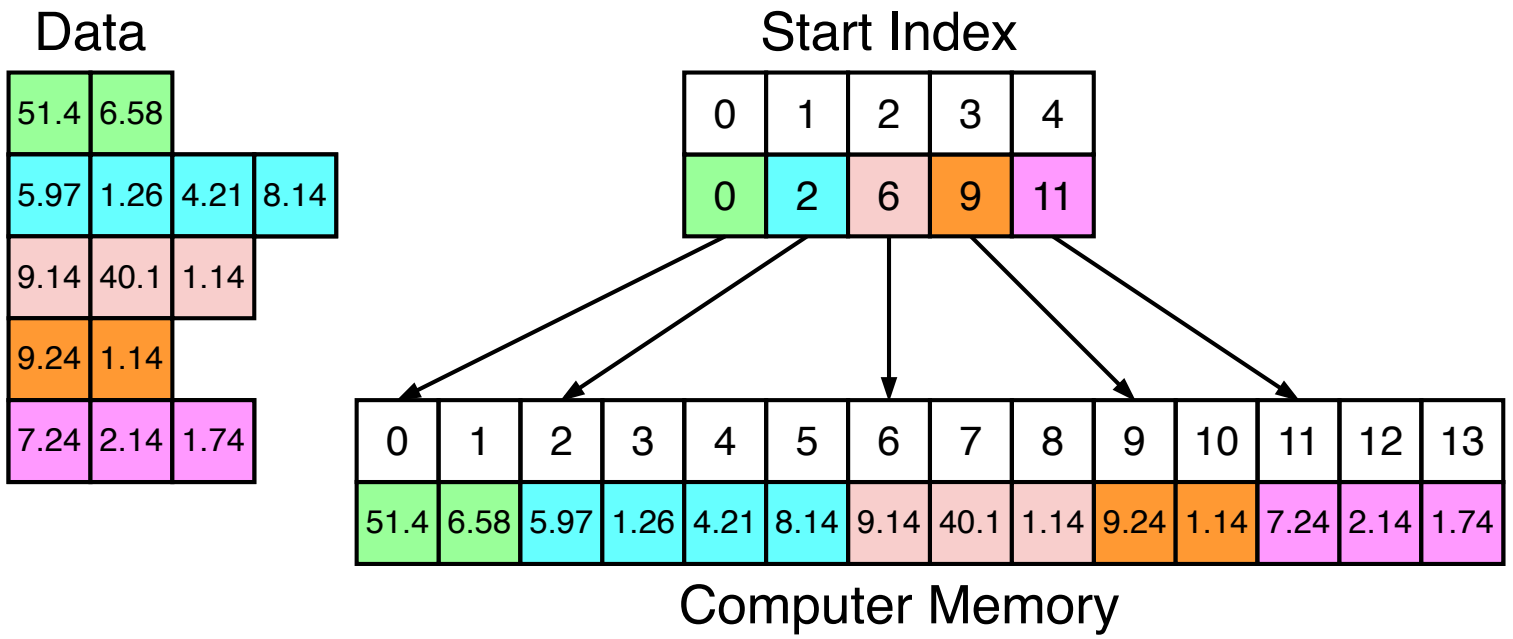
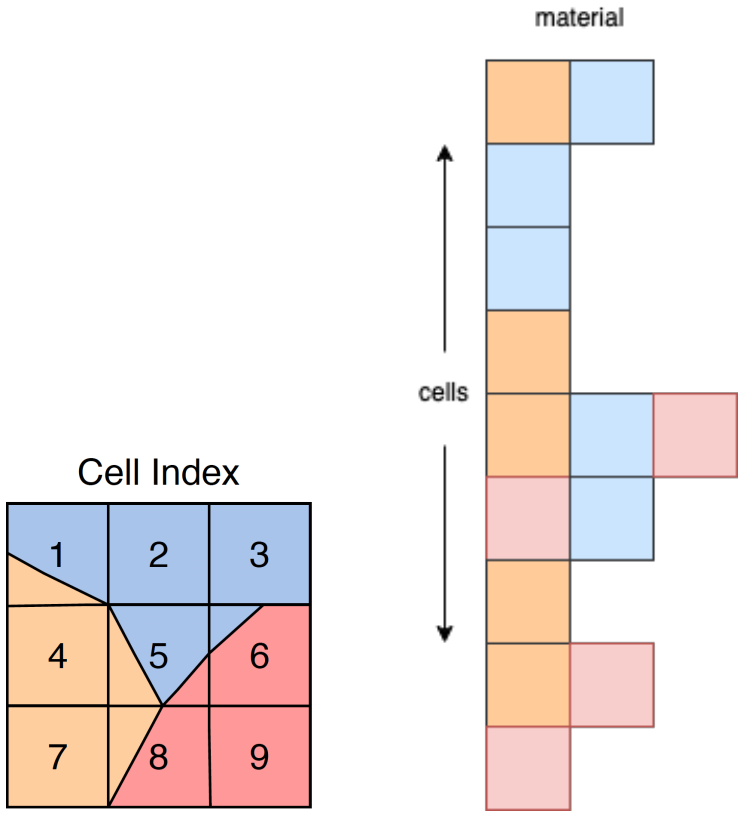


Figure describing the layout of a ragged-right -i.e the columns for each row vary. The user accesses the data as a 2D array, `(i,j)`, but the data in memory is stored as a contiguous 1D array by rows.

Ragged Right Example

Building a Ragged-Right for the cell centric multi-material matrix example. The cell information is contiguous in memory.Focus is the material in each cell.



- Create the array: `auto multimat_rr = RaggedRightArray<double>(row_length_arr, row_dim)`
- Accessing data: `multimat_rr(i,j)`

MATAR Sparse Data Structure: Ragged-Down

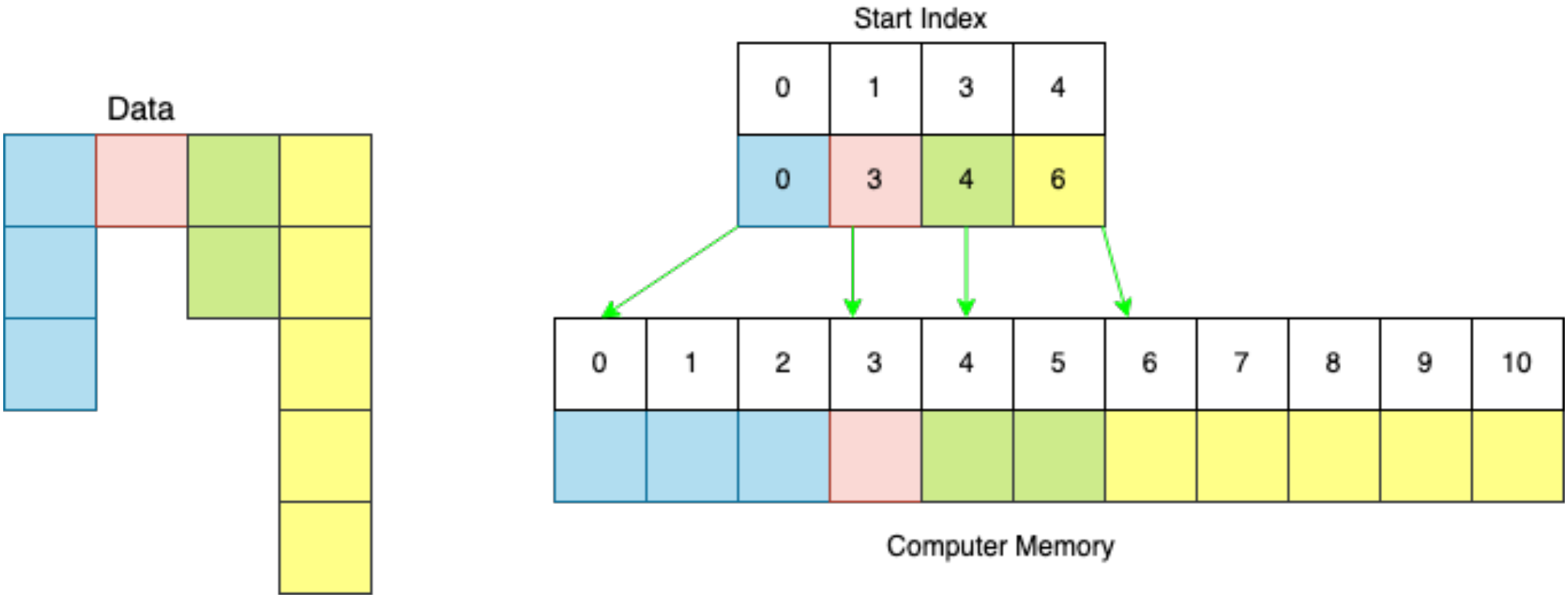
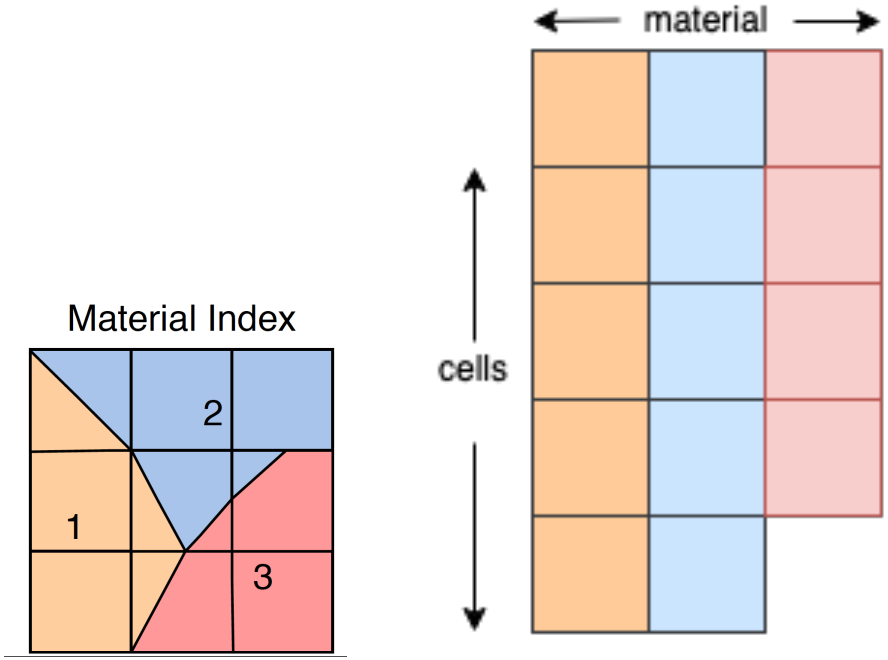


Figure describing the layout of a ragged-down -i.e the rows for each column vary. The user accesses the data as a 2D array, `(i,j)`, but the data in memory is stored as a contiguous 1D array by columns.

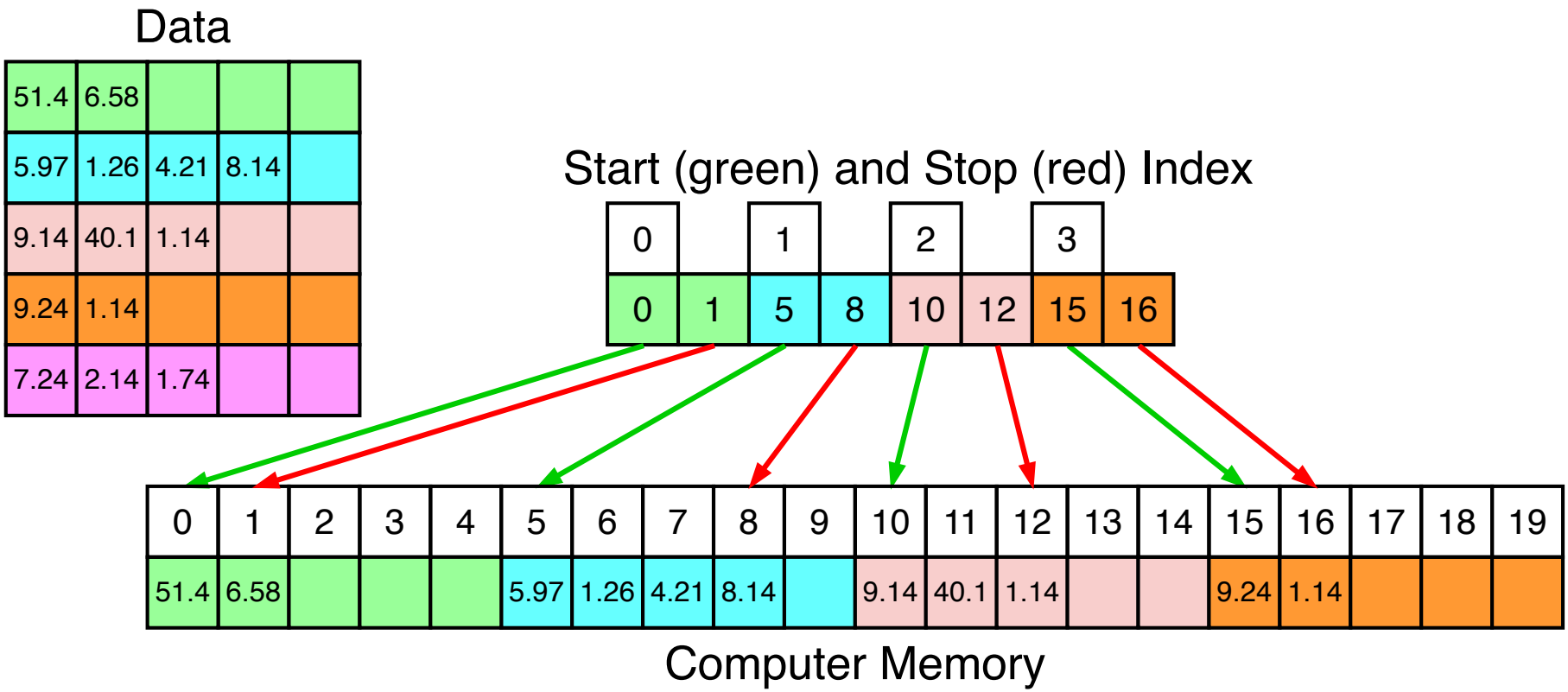
Ragged Down Example

Building a Ragged-Down for the material centric multi-material matrix example. The data is contiguous by material. Focus is the cells that containg the material.



- Create the array: `auto multimat_rr = RaggedRightDown<double>(col_length_arr, col_dim)`
- Accessing data: `multimat_rr(i,j)`

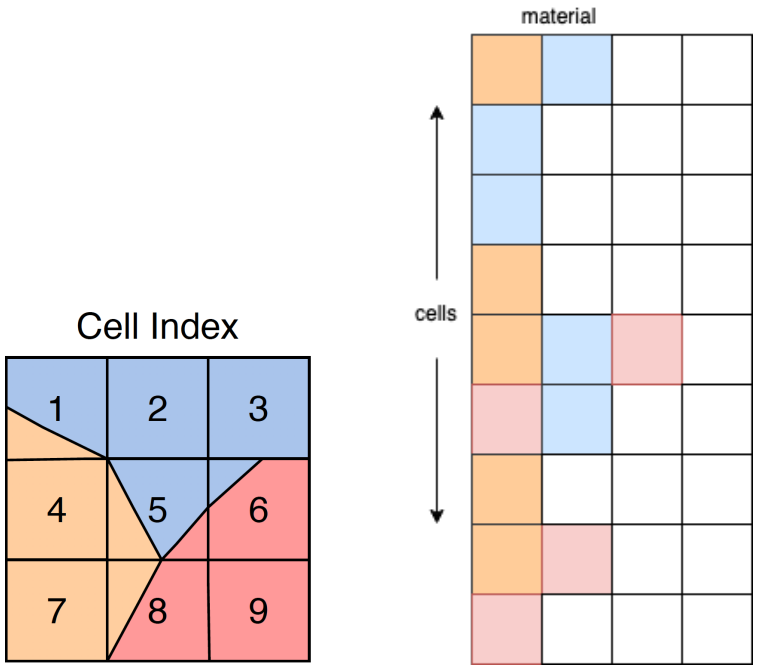
MATAR Sparse Data Structure: Dynamic Ragged-Right



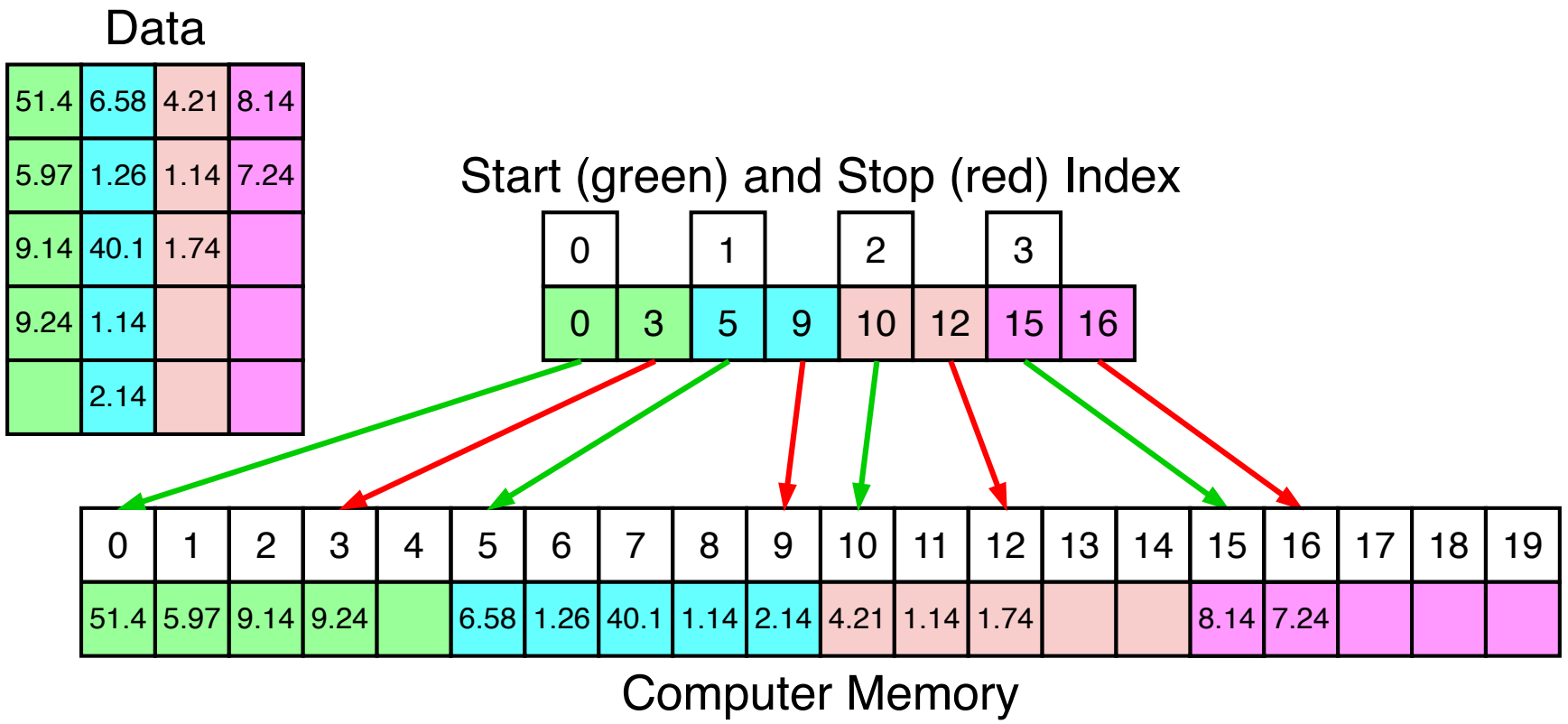
Dynamic ragged-right has memory buffers available in order to add more entries at the end of each row.

Dynamic Ragged Right Array Example

Multi-material matrix with dynamic ragged arrays. White boxes are buffers, columns vary but the rows are a fixed size. The information for each cell is contiguous in memory.



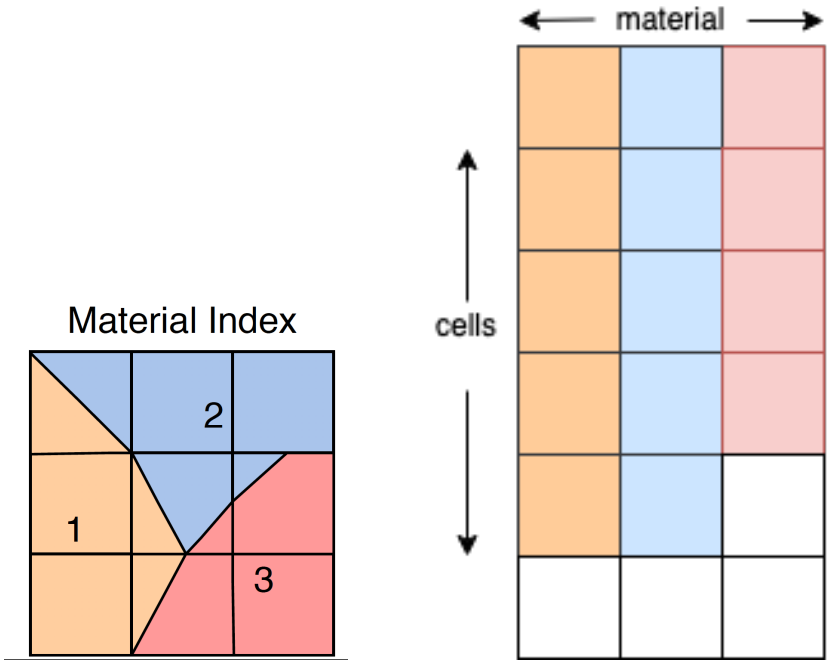
MATAR Sparse Data Structure: Dynamic Ragged-Down



Describing the dynamic ragged-down array -i.e the rows of each column vary. Buffer to add more data at the end of each column.

Dynamic Ragged Down Array Example

Multi-material matrix with dynamic ragged arrays. White boxes are buffers, rows vary but the columns are a fixed size. The material is stored contiguous in memory.



Sparse Data Representation Summary

1. `RaggedRight` and `RaggedDown`
   - Both dimensions are `fixed` when created. i.e how many columns each row of the `RaggedRight` array is known and unchanged when created..
   - Varying number of rows/columns.

Sparse Data Representation Summary

1. `RaggedRight` and `RaggedDown`
   - Both dimensions are `fixed` when created. i.e how many columns each row of the `RaggedRight` array is known and unchanged when created..
   - Varying number of rows/columns.
2. `DynamicRaggedRight` and `DynamicRaggedDown`
   - *One* of the dimensions is fixed (either rows or columns).
   - Data can only be added at the buffers, i.e it will be stored at the end of the row/column.
   - Addresses the contiguous data layout issue of linked list, but does not have the other insertion/deletion properties.

Application Code: HOSS

About Hybirid Optimization Software Suite (HOSS):

- HOSS is a geophysical application developed at LANL.
- The software integrates solid mechanics using finite-discrete element method and computational fluid dynamics.
- Simulate deformation and failure of material with applications to oil, gas, mining and defense.
- Provides a real-world test for the technique in a multi-physics production application.

Application Code: HOSS

About Hybirid Optimization Software Suite (HOSS):

- HOSS is a geophysical application developed at LANL.
- The software integrates solid mechanics using finite-discrete element method and computational fluid dynamics.
- Simulate deformation and failure of material with applications to oil, gas, mining and defense.
- Provides a real-world test for the technique in a multi-physics production application.

MATAR + HOSS

Use data from a contact detection algorithm to time adding elements, traversing, and simple computations of a linked list and a `dynamic ragged-right`.

Linked List vs. Dynamic Ragged-Right

From the contact detection data:

- Every element within the simulation has it's own linked list containing neighbours.

Linked List vs. Dynamic Ragged-Right

From the contact detection data:

- Every element within the simulation has it's own linked list containing neighbours.
- As the simulation progress, nodes are added if elements come into contact. Removed if no longer in contact.

Linked List vs. Dynamic Ragged-Right

From the contact detection data:

- Every element within the simulation has it's own linked list containing neighbours.
- As the simulation progress, nodes are added if elements come into contact. Removed if no longer in contact.
- Size of test problem: 10 million elements.

Linked List vs. Dynamic Ragged-Right

From the contact detection data:

- Every element within the simulation has it's own linked list containing neighbours.
- As the simulation progress, nodes are added if elements come into contact. Removed if no longer in contact.
- Size of test problem: 10 million elements.
- Two test were conducted:
  - One iteration of the contact detection to evaluate time it takes to build the data structure.
  - Quantify difference in speed between the two structures while being used.

Linked List vs.Dynamic Ragged-Right: Building the Structure

Initial performance test: building and initializing the data structure.

Performance Metric	LL	DRRA
Run time [s]	16.17	15.70
DP MFLOP/s	119.98	123.72
AVX DP MFLOP/s	0.76	0.85

Table: LIKWID results for the building and initializng of the data structures, showing that the DRRA is about 3% faster.

Linked List vs.Dynamic Ragged-Right: Quantify Speed Difference

To simulate production run, row statistics (row avg, min, max, sum) were calculated over each row for 10k iterations.

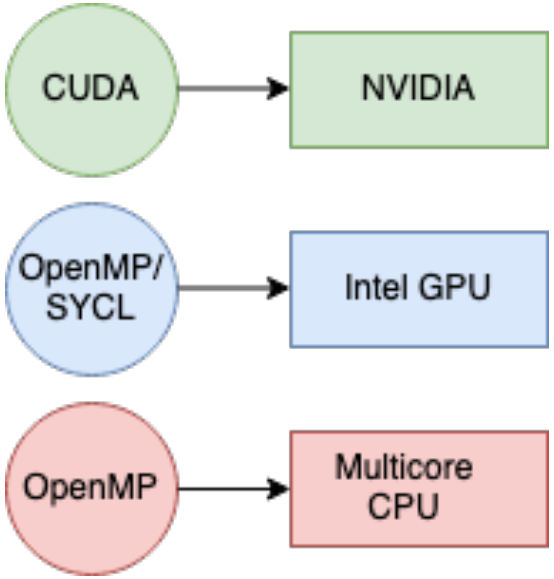
Performance Metric	LL	DRRA
Runtime [s]	580.47	325.67
DP MFLOP/s	453.95	568.56
AVX DP MFLOP/s	0	0

Table: LIKWID results on competing HOSS implementation. We see a drastic improvement in runtime with the DRRA over the linked list. We attribute the speed up to the contiguous memory access, which minimizes the amount of memory that has to be loaded.

Portability of MATAR to GPU

Focus on Portability

GPU programming has become more popular for large scale computing. Our focus is designing the data structures to be portable across various architectures.



MATAR + Kokkos

- Kokkos is a C++ parallel programming portability library.
- Uses OpenMP, CUDA, PThreads as backend programming models.
- MATAR uses Kokko's Views as a backend in order to represent the dense and sparse arrays on the GPU.

MATAR + Kokkos = sparse and dense data representationon GPU

Dense Datatype + Kokkos

Tested if there is any overhead using Kokko's datatypes as a backend using Stream Benchmark.

Data type	Copy	Scale	Sum	Triad	Dot. Prod
Kokkos View	6.57	6.49	8.51	9.00	7.08
CArray	6.56	6.26	8.16	8.26	6.37

Table: Comparison of 3D MATAR CArray and regular Kokko's View. Dimension is $256 \times 256 \times 256$ on IBM Power 9 CPU using OpenMP. Units are in milliseconds

Dense Datatype + Kokkos

Tested if there is any overhead using Kokko's datatypes as a backend using Stream Benchmark.

Data type	Copy	Scale	Sum	Triad	Dot. Prod
Kokkos View	6.57	6.49	8.51	9.00	7.08
CArray	6.56	6.26	8.16	8.26	6.37

Table: Comparison of 3D MATAR CArray and regular Kokko's View. Dimension is $256 \times 256 \times 256$ on IBM Power 9 CPU using OpenMP. Units are in milliseconds

Data type	Copy	Scale	Sum	Triad	Dot. Prod
Kokkos View	0.41	0.41	0.49	0.49	1.69
FArray	0.45	0.45	0.50	0.50	1.83

Table: Comparison of 3D MATAR FArray and regular Kokko's View. Dimension is $256 \times 256 \times 256$ on NVIDIA V100 GPU. Units are in milliseconds

Matrix-Matrix Multiply Portability

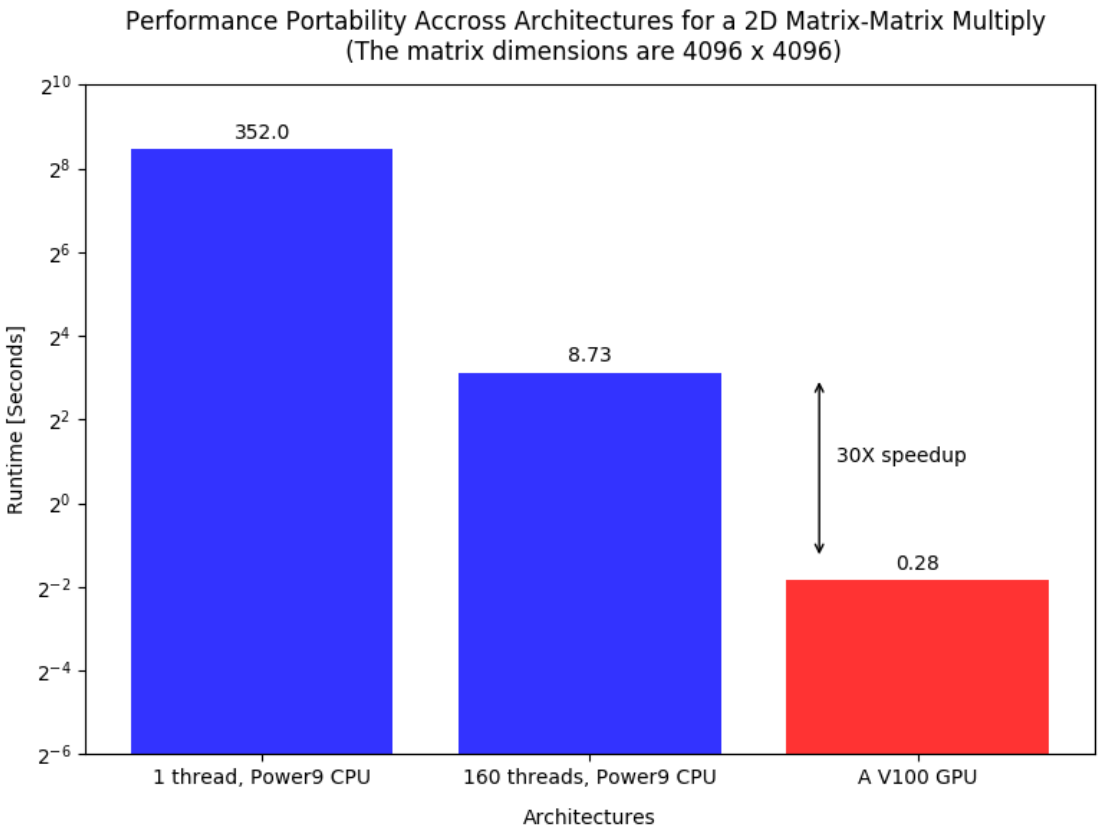


Figure showing the scaling for MM across serial, CPU parallel and GPU parallel.

- Linear scaling is observed on the multi-core CPU.
- Show portability of the problem; significant speed up with minimal overhead by using `Kokko's Views`.

Conclusion

- We present MATAR: a portable, performant C++ library that can for both dense and sparse data representation.
- We are able to use `Kokkos` without significant any overhead on the dense data structures.
- Speed up on common linear algebra algorithms and on a contact detection algorithm HOSS.

References

1. MATAR: https://github.com/lanl/MATAR ▸ Link
2. Boost: https://www.boost.org ▸ Link
3. HOSS: ▸ Link
4. Warwich Data Store: R.O. Kirk, M, Nolten, R. Kevis, T.R. Law, S. Mahenswaran, S.A. Wright, S. Powell, G.R. Mudalige, S.A Jarvis, Warwick data store: A data structure abstraction library. in 2020 IEEE/ACM Performance Modeling, Benchmarking and Simulation of High Performance Computer Systems.
5. Matrix Template Library: J. Siek, A. Lumsdaine, The matrix template library: A generic programming approach to high performance numerical linear algebra.
6. Multimaterial study: S. Fogerty, el al. A comparative study of multi-material data structures for computational physics applications. Computer and Mathematics with Applications (2018).